

# 基于分层模型的组合服务故障定位算法

崔鹏帅,朱浩洋,任 浩

(国防科学技术大学 计算机学院,湖南 长沙 410073)

**摘 要:**在面向服务的架构中,服务之间的依赖关系具有单向性的特点。基于这种单向性依赖,提出了服务故障传播的分层模型并设计了服务的分层算法,将服务节点分层。根据分层模型设计了服务故障定位的监测探针和诊断部署,减少了监测探针的数目。在探针探测结果的基础上,提出了分层模型下基于贝叶斯网络的故障定位算法,该算法通过计算故障发生时服务的影响因子和可信度,快速定位故障。仿真结果验证了该算法可以较准确地定位组合服务中的故障,且保持较低的误报率。

**关键词:**网络服务;故障定位;分层模型;贝叶斯网络

中图分类号:TP301.6

文献标识码:A

文章编号:1673-629X(2014)09-0006-05

doi:10.3969/j.issn.1673-629X.2014.09.002

## A Fault Location Algorithm of Composition Service Based on Hierarchical Model

CUI Peng-shuai, ZHU Hao-yang, REN Hao

(College of Computer, National University of Defense Technology, Changsha 410073, China)

**Abstract:** The dependencies between services have the property of unidirection in Service-Oriented Architecture (SOA). A hierarchical model of services fault propagation is proposed based on the property and a hierarchical algorithm is designed to classify the services into different layers. Active probing approach is also used to detect the service's symptom, and the number of detection probing has been decreased as a result of the use of hierarchical model. Based on the result of detection probing, a fault location algorithm under the hierarchical model based on Bayesian network is put forward, which can locate the fault service quickly by calculating the impact factor of every service when there exist faults. The simulation results show that the fault location algorithm can accurately locate service fault of composition Web Service and maintain a low rate of false positive.

**Key words:** Web Service; fault location; layering model; Bayesian network

## 0 引言

传统的软件开发缺少组件间的复用,造成开发成本上升、软件紧耦合、灵活度下降等问题。面向服务的架构(Service-Oriented Architecture, SOA)为一种组件的复用技术被引入到软件开发中来<sup>[1]</sup>。SOA具有松耦合、可复用、基于协议、黑盒等特性,在其解决软件开发中的紧耦合和不可复用等问题时,监管问题也随之而来。服务实现对外部透明、服务可根据功能需求随意组合和复用等特性使得服务管理的成本不断上升,管理效率却不断下降。

服务故障管理是服务管理的重要组成部分,而故

障定位是故障管理的核心部分。目前,在故障探测手段上,主动探针技术被越来越多地运用。探针早期主要用于网络层的故障诊断,Natu等<sup>[2]</sup>将其拓展到整个网络的故障诊断,并设计了不同功能的探针。褚灵伟等<sup>[3]</sup>进一步将其应用到网络服务的故障诊断中来,并通过定义探针的监测能力设计了探针选择算法,然而其探针选择算法复杂度较高,而且在每次服务依赖关系的变化都需要重新计算并选取探针。在故障定位算法方面,Lin<sup>[4]</sup>提出了基于模型的故障定位算法,其主要优点是在服务节点无法确定先验概率的情况下,基于模型的故障定位算法仍然可以定位故障,然而其主要缺点是模型的构建需要系统的实际结构,对于难以

收稿日期:2013-09-05

修回日期:2014-01-03

网络出版时间:2014-05-21

基金项目:国家自然科学基金资助项目(61170285)

作者简介:崔鹏帅(1990-),男,河南安阳人,硕士研究生,CCF会员,研究方向为服务管理、信息安全;任 浩,副教授,博士,研究方向为网络应用。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20140525.1242.001.html>

构建模型的问题无能为力,而且模型的构建十分复杂和困难。Huang 等<sup>[5]</sup>采用最大覆盖算法进行故障定位,其核心思想是用同时发生故障的概率很小,因此用最小的故障子集覆盖症状集,该故障子集即是诊断结果。但是在实际情况中,大的故障子集发生的概率可能大于小的故障子集。张顺利等<sup>[6]</sup>提出了虚拟环境下的服务故障诊断算法,并在算法中引入可信度机制,但是未考虑服务本身的故障先验概率。Liao 等<sup>[7]</sup>提出了通行网络中的近似故障定位算法,降低了故障定位的时间复杂度,故障诊断的准确率接近穷举算法,但是未考虑 SOA 架构下服务的故障定位。李晶等<sup>[8]</sup>提出了基于事件驱动的 SOA 环境下的故障定位算法,将故障疑似集的选择转换为  $k$ -median 问题。杜晓丽等<sup>[9]</sup>设计了基于依赖图的故障定位算法,然而其算法需要迭代较多的次数才能定位故障,不适合服务节点较多和服务依赖较复杂的网络。范贵生等<sup>[10]</sup>提出了基于 Petri 网的服务组合故障诊断与处理方法,可以对不同的服务故障类型进行建模,但该方法更多地关注于服务故障的处理过程和服务组合的成功率,若不能重新组合,则很难定位具体故障源并进行处理。

## 1 服务分层模型

### 1.1 分层模型的建立

文中的服务分层是将服务根据其依赖关系划分层次,划分层次的作用有以下几点:

- (1) 确定组合服务的顶层服务,作为周期性探针,即监测探针的探测节点;
- (2) 确定服务的层次结构,在服务依赖关系发生变化时能及时分层,确定新的顶层服务;
- (3) 在服务故障定位后,能根据服务的具体层次直观地确定故障的影响因子;
- (4) 分层的结构能为知识库的快速更新提供便利。

服务分层后的结构如图 1 所示。

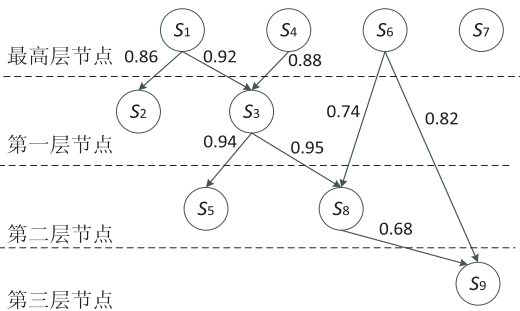


图 1 服务分层结构

在服务分层的基础上,构建了故障传播的分层模型。为描述该模型,先作如下定义:

定义 1: 服务集合  $S$ ,  $S$  包括所有已在注册中心注

册的服务及其性质。 $S = \{S_1, S_2, \dots\}$ , 其中  $S_1, S_2, \dots$  等分别为单个服务,  $S$  中服务的个数表示为  $|S|$ 。

定义 2: 服务边序  $R_i$ , 如果服务  $S_1$  直接依赖服务  $S_2$ , 即  $S_2$  的输出结果作为  $S_1$  的输入参数, 则存在边序  $R_i = \langle S_1, S_2 \rangle$ 。

定义 3: 依赖关系集合  $R$ , 即服务间所有依赖关系。 $R = \{R_1, R_2, \dots\}$ ,  $R$  中关系的个数表示为  $|R|$ 。

定义 4: 服务层数集合 Layer, 即标明分层后服务所在层数,  $\text{Layer} = \{S_1. \text{Layer}, S_2. \text{Layer}, \dots\}$ , 服务  $S_i$  所在层表示为  $S_i. \text{Layer}$ , 所有服务的最大层数表示为  $\text{MaxLayer}$ 。

定义 5:  $P(F_{S_i})$  为服务  $S_i$  出现故障的先验概率,  $P(F_{S_i})$  的值可由观测直接给出, 也可以根据诊断的历史数据实时更新。

定义 6:  $P(N_{S_i})$  为服务出现负症状的概率,  $P(N_{S_i})$  的值可由观测直接给出, 也可根据服务的故障概率和关联关系计算得出。

定义 7:  $P(N_{S_i} | F_{S_j})$  为直接关联因子, 即服务  $S_j$  出现故障时  $S_i$  直接表现出症状的概率, 该值与通信的传输信道和依赖类型相关, 可以先根据经验给出, 也可用故障注入技术<sup>[11]</sup>获取。

定义 8:  $\text{Child}(S_i)$  为与服务  $S_i$  存在因果关系的低层节点, 即所有发生故障会使得  $S_i$  产生症状的节点。

定义 9:  $\text{Parent}(S_i)$  为与服务存在因果关系的高层节点, 即该服务发生故障时所有可能会产生症状的节点。

根据分层模型的依赖关系, 将故障传播模型建立为以故障树组成的森林, 将其形式化定义为  $\text{FF} = (S, R, K)$ ,  $S$  为已经在注册中心注册过的服务集合,  $R$  为注册时填写的或通过其他手段获取的依赖关系集合,  $K$  为故障诊断所依赖的其他知识, 主要包括服务故障的先验概率, 服务层数 Layer, 服务之间的关联因子  $P(N_{S_i} | F_{S_j})$ , 各个服务的高层节点  $\text{Parent}(S_i)$  和服务的低层节点  $\text{Child}(S_i)$ 。

目前, 对服务故障分类已有较多研究<sup>[12-13]</sup>, 文中将目光集中在定位算法中, 仅把服务分为无故障和有故障两种类型, 将症状分为正症状和负症状, 其中正症状为探测服务状态为无故障, 负症状为探测服务状态为有故障。故障传播时, 因为不可靠的网络等影响, 症状在传播中会出现症状丢失或虚假症状等情况, 因此症状的传播是一个概率事件, 然而高层服务的每次执行都需要获取低层服务数据, 在多次执行后, 低层服务的症状总能传播到高层。因此, 只需关注服务的最高层节点, 就能感知出所有的服务中是否出现了症状。如在图 1 中, 服务  $S_8$  出现故障会使得服务  $S_8, S_3, S_1$ ,

$S_6, S_7$  出现症状,因此在图 1 中只需监测  $S_4, S_1, S_6, S_7$  就能感知所有服务是否出现症状。

## 1.2 服务分层算法

服务分层结构需要将服务分层,因此设计了如下的服务分层算法。

算法 1: 服务分层算法。

输入: 服务间的依赖关系  $R$ , 服务集合  $S$ ;

输出: 服务层数  $Layer$ 。

Step1: 获取  $R$  中关系的个数  $R.length = |R|$

Step2: for (int  $i=0; i < R.length; i++$ )

```
{
for (int  $j=0; j < R.length; j++$ )
```

```
{
在  $S$  中寻找  $R_i$  的第一个节点  $s_m$ ;
在  $S$  中寻找  $R_i$  的第二个节点  $s_n$ ;
```

```
if ( $s_m.layer > s_n.layer$ )
 $s_n.layer = s_m.layer + 1$ ;
```

```
}
}
```

Step3: 服务层数集合  $Layer$

## 1.3 服务分层更新算法

在对服务进行分层后,为确保在故障定位中能实时得到最新的分层信息,需要对服务的层数不断更新。解决方案之一是设置一个时间间隔  $T$ ,每隔  $T$  就重新运行服务的更新算法,然而在服务节点数较多的情况下,每一次的更新都需要浪费大量时间,因此需要根据服务依赖关系的动态变化来更新服务层数。

算法 2: 服务分层更新算法。

输入: 服务集合  $S$ , 服务关系集合  $R$ , 服务层数集合  $Layer$ , 服务关系变化  $Change$ ;

输出: 更新后的服务层数集合  $Layer$ 。

Case One:  $Change = S$  中新加入一个服务  $s_{new}$

Step1:  $s_{new}.Layer = 0$

Step2: return  $S$

Case Two:  $Change = R$  中新加入一个依赖关系  $<$

```
 $s_{exist_1}, s_{exist_2}>$ 
refresh_add( $<s_{exist_1}, s_{exist_2}>$ )
```

```
{
Step1: if ( $s_{exist_1}.Layer > s_{exist_2}.Layer$ )
```

```
 $s_{exist_2}.Layer = s_{exist_1}.Layer + 1$ 
for all  $exist_{<s_{exist_2}, s_{exist_1}>}$  in  $R$ 
refresh_add( $<s_{exist_2}, s_{exist_1}>$ )
```

Step2: return  $S$

```
}
```

Case Three:  $Change = R$  中删除一个依赖关系  $<$

```
 $s_{exist_1}, s_{exist_2}>$ 
```

```
refresh_delete( $<s_{exist_1}, s_{exist_2}>$ )
```

```
{
```

Step1: int  $buf = s_{exist_2}.Layer$

```
 $s_{exist_2}.Layer = 0$ 
```

Step2: for all  $exist(<s_{exist_1}, s_{exist_2}>)$

```
if ( $s_{exist_1}.Layer > s_{exist_2}.Layer$ )
```

```
 $s_{exist_2}.Layer = s_{exist_1}.Layer + 1$ 
```

Step3: if ( $buf \neq s_{exist_2}.Layer$ )

```
for all  $exist(s_{exist_2}, s_{exist_1})$ 
```

```
refresh_delete( $s_{exist_2}, s_{exist_1}$ );
```

Step4: return  $S$

```
}
```

Case Four:  $Change = S$  中删除一个服务  $s_{exist_1}$

Step1: for all  $exist <s_{exist_1}, s_{exist_1}>$  in  $R$ ;

```
refresh_delete( $<s_{exist_1}, s_{exist_1}>$ );
```

Step2: for all  $exist <s_{exist_2}, s_{exist_1}>$  in  $R$ ;

```
refresh_delete( $<s_{exist_2}, s_{exist_1}>$ )
```

## 1.4 探针部署设计

把探针分为 2 类,监测探针和诊断探针。监测探针平时监测整个系统是否存在故障或症状,当系统中服务有故障发生出现症状时,至少有一个监测探针能探测到有服务存在症状。在服务分层的情况下,只需监测所有顶层服务,因此监测探针的数目即为顶层服务的数目。诊断探针在监测探针发现症状后,探测出现症状的服务的所有下层服务,即所有出现症状的服务的  $Child(s)$  为故障定位提供数据。

为部署诊断探针,需要计算每个服务的下层相关服务  $Child(s)$ ,在服务已经分层的情况下,算法如下:

算法 3: 分析服务相关低层服务。

输入: 服务集合  $S$ , 服务关系集合  $R$ , 服务层数集合  $Layer$ ;

输出: 所有服务的相关低层服务  $Child(s)$ 。

```
for (int  $i = MaxLayer; i > 0; i--$ )
```

```
{ for (int  $k=0; k < |S|; k++$ )
```

```
{ if ( $s_k.Layer = i$ )
```

```
 $s_k.downlist += s_k$ ; //downlist 为  $s_k$  的相关下层服务集合
```

```
for (int  $l=0; l < |R|; l++$ )
```

```
{ if ( $R_l = <s_k, s_n>$ )
```

```
 $s_k.downlist += s_n.downlist$ ; //合并集合
```

```
}
```

```
 $Child(s_k) = s_k.downlist$ 
```

```
}
```

```
}
```

和服务分层更新算法类似,也可以设计服务相关低层服务更新算法,这里不再赘述。

## 2 故障定位算法

### 2.1 故障定位算法简介

在上一节中,用分层算法将服务分层,用分层的更新算法实时更新服务的分层结构,并构建了故障传播的分层模型。在系统中有故障发生时,监测系统捕捉到症状,并通过故障定位算法定位故障源。为描述故障定位算法,作如下定义:

定义 10:疑似故障节点权值  $C(s_i | S_y)$  为症状  $S_y = \{S_{y_1}, S_{y_2}, \dots\}$  发生时服务节点  $S_i$  的贡献度,  $C(s_i | S_y)$  的计算如公式(1)所示,其中  $P(s_i | S_{y_j})$  表示症状  $S_{y_j}$  由服务节点  $S_i$  引起的概率。

$$C(s_i | S_y) = \sum_{j=1}^n P(s_i | S_{y_j}) \quad (1)$$

根据贝叶斯定理

$$P(s_i | S_{y_j}) = P(F_{s_i}) \times P(N_{s_j} | F_{s_i}) / P(N_{s_j}) \quad (2)$$

定义 11:疑似故障节点可信度为节点  $S_i$  被确定为疑似故障节点的可信度,其取值范围为  $[0, 1]$ ,其中 0 表示完全不可信,1 表示为完全可信。

$$\gamma_i = |S_{y_{s_i \rightarrow S}} \cap S_y| / |S_{y_{s_i \rightarrow S}}| \quad (3)$$

定义 12:可信度下疑似故障节点权值,即考虑可信度的情况下,出现症状  $S_y$  时服务节点的故障权值。

$$V(s_i) = C(s_i | S_y) \times \gamma_i \quad (4)$$

在式(2)中,  $P(N_{s_j})$  可以按照如下方法给出,设  $H_{s_j} = \{s_1, s_2, \dots, s_m\}$  为所有发生故障会导致服务  $s_j$  出现症状的服务集合,则  $P(N_{s_j})$  的计算如公式(5)所示。

$$P(N_{s_j}) = 1 - \prod_{i=1}^m (1 - P(F_{s_i}) \times P(N_{s_j} | F_{s_i})) \quad (5)$$

其中,  $P(F_{s_i})$  和  $P(N_{s_j} | F_{s_i})$  的值可以预先假设,也可根据历史统计信息获得,其值可以存放在知识库  $K$  中,定位时可随时取出,定位结束后,可以根据定位结果进行适当修正;在式(3)中,表示所有可能由服务  $S_i$  故障引起的症状集合,即  $\text{Parent}(S_i)$ ,其计算和更新与  $\text{Child}(s)$  类似,这里不予赘述。 $S_y$  为观测到的负症状集合。

故障定位的目标是找出最大概率的故障集,使得该故障集传播的症状能覆盖所有探针探测到的症状集,需要说明的是,在文中算法中定位的最大概率故障集不一定是能解释所有症状的故障集的最小集合。

算法 4:故障定位算法。

输入:故障传播模型  $FF=(S, R, K)$ , 探测得到的症状集  $S_y$ ;

输出:故障集  $F$ 。

Step1:置故障集  $H$  为空集;

Step2:计算所有节点的  $V(s_i)$ ;

Step3:选出拥有最大的  $V(s_i)$  的节点,将  $s_i$  加入故障集  $H$ ;对于所有使得  $P(s_i | S_y)$  不为 0 的  $s_j$ ,  $S_y = S_y - s_j$ ;

Step4:重复步骤 1,2 直至  $S_y$  为空集;

Step5: $H$  即为定位的故障集。

### 2.2 间接关联方法下的故障定位

在 2.1 中,故障定位算法中所使用的关联关系都是服务注册时或根据其 WSDL (Web Services Description Language) 描述得出的直接关联关系,设想如下场景,服务依赖关系和先验故障概率如图 2 所示。当  $S_0$  出现故障时,  $S_0 \sim S_3$  均出现症状,根据 2.1 节权值计算和算法,得出最大权值为  $S_1$ ,将  $S_1$  加入故障集,在症状集中删除  $S_1 \sim S_3$ ,具有最大权值的是  $S_0$ ,将  $S_0$  加入故障集,因此故障定位的结果是  $\{S_0, S_1\}$ 。

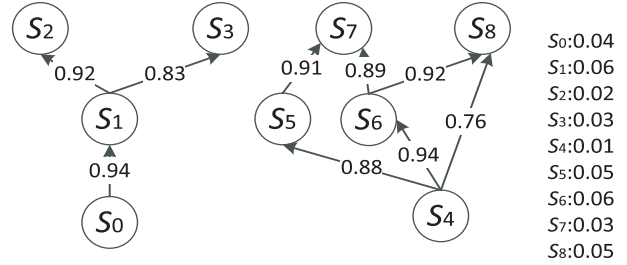


图2 服务关联示例

在上述场景下出现误报率的主要原因是使用直接关联关系时,故障定位表现出比较强的局部性,即选出的最大权值服务节点与当前症状直接相关,没有考虑症状的原因是否为更下层故障传,为此,考虑用间接关联关系替代直接关联关系进行故障定位,故障定位算法与使用直接关联因子相同,下面给出间接关联因子的定义。

定义 12:间接关联因子,  $P_{in}(N_{s_i} | F_{s_j})$  即下层服务发生故障或症状时导致上层服务  $s_i$  出现症状的概率。

$P_{in}(N_{s_i} | F_{s_j})$  的计算如下:若  $s_i$  与  $s_j$  具有直接关联关系,则经过服务  $s_j$  的间接概率  $P_{in}(N_{s_i} | F_{s_j})_{s_j} = P_{in}(N_{s_i} | F_{s_j}) P_{in}(N_{s_i} | F_{s_j})_{s_j}$ ;若服务  $s_i$  与  $s_k$  的间接关联因子为  $P_{in}(N_{s_i} | F_{s_i}) = p_1$ ,服务  $s_k$  与  $s_i$  的间接关联因子为  $P_{in}(N_{s_i} | F_{s_j}) = p_2$ ,则经由服务的间接关联因子为  $P_{in}(N_{s_i} | F_{s_j})_{s_k} = p_1 \times p_2$ ;若当前已知服务  $s_i$  与  $s_j$  的间接关联因子为  $P_{in}(N_{s_i} | F_{s_i}) = p_1$ ,新发现通过服务的路径  $P_{in}(N_{s_i} | F_{s_j})_{s_k} = p_2$ ,则更新后的间接关联因子为  $P_{in}(N_{s_i} | F_{s_j}) = p_1 = 1 - (1 - p_1) \times (1 - p_2)$ 。

如在图 2 中,  $P_{in}(N_{s_2} | F_{s_0}) = P_{in}(N_{s_2} | F_{s_1}) \times P_{in}(N_{s_2} | F_{s_1})_{s_0}$ ,最终值为  $0.8648$ ;  $P_{in}(N_{s_8} | F_{s_4})_{s_6} = 0.92 \times 0.94 = 0.8648$ ;  $P_{in}(N_{s_8} | F_{s_4})_{s_4} = 0.76$ ,则  $P_{in}(N_{s_8} | F_{s_4}) = 1 - (1 - 0.76) \times (1 - 0.84)$ ,最终值为  $0.9676$ 。

服务分层后,可以很容易地从低层到高层逐层计算,即先计算最底层服务与上层所有服务的间接关联



关系,再计算次低层与其上层所有服务的间接关联关系。而且分层后,可以根据服务依赖的动态变化动态地更新间接关联因子。因为篇幅限制,间接关联因子的计算算法和更新算法这里不做详细阐述。

### 3 仿真结果及分析

#### 3.1 实验环境

文中采用 BNGenerator<sup>[14]</sup> 产生的服务网络作为测试数据,测试的服务规模为 10~120。在相同服务规模下,不同结构的贝叶斯网络会直接影响实验结果,为保证数据的真实可靠,对每个相应规模的测试采用随机产生 100 个不同的贝叶斯网络,在每个贝叶斯网络中模拟 10 000 次,最后取平均值作为该规模下的测试结果。

#### 3.2 评价指标

文中采取与文献[3]和文献[6]相似的评价指标,分别为监测探针数目、诊断探针数目、诊断率(Diagnosis Accuracy, DA)、误判率(False-Positive, FP)。其中 DA 和 FP 定义如下:

$$DA = |F_T \cap H| / |F_T| \quad (6)$$

$$FP = |H - F_T| / |H| \quad (7)$$

其中,  $F_T$  为真实发生的故障;  $H$  为故障定位算法得到的故障集。

#### 3.3 结果分析

图 3 和图 4 统计了不同服务规模下探针的数目,其中监测探针为周期性探针,用于探测整个系统是否出现症状;诊断探针为实时探针,用于诊断故障。在仿真结果中,监测探针和诊断探针和服务规模成正比关系。在实际应用中,监测探针要保持周期性探测,诊断探针则为出现症状时才使用,文中算法保持了较小的监测探针数目,降低了整个监控中的探测干扰度。

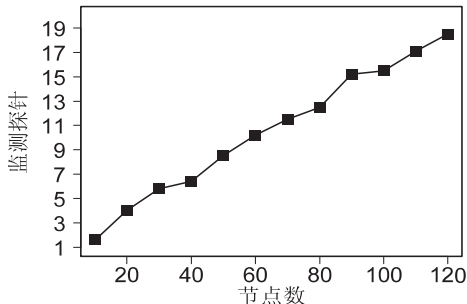


图 3 监测探针数

诊断率的结果如图 5 所示,图中直接关联曲线和间接关联曲线分别表示了采用直接关联法进行故障诊断和间接关联法进行故障诊断的准确率。直接关联法每次都根据直接关联结果进行故障定位,注重局部诊断的准确性;而间接关联法注重全局的诊断的准确性,因此直接关联法故障集大于间接关联法故障集,其准

确度高于间接关联法,误判率也高于间接关联法,结果如图 6 所示。

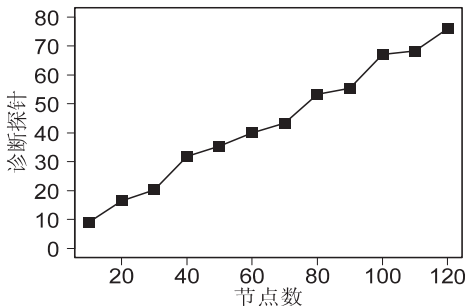


图 4 诊断探针数

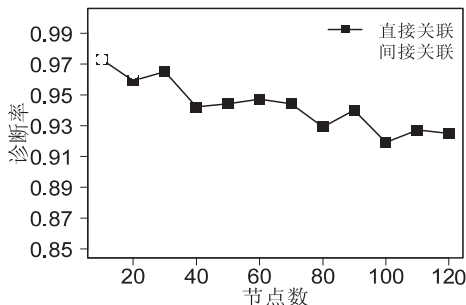


图 5 故障诊断率

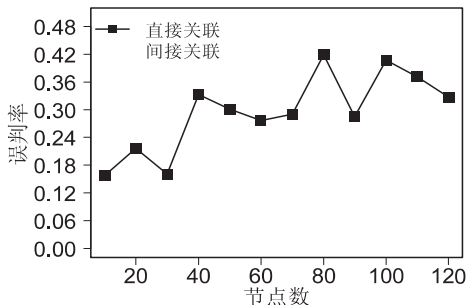


图 6 故障误判率

### 4 结束语

SOA 架构是解决软件松耦合的重要方法,但 SOA 的特性给服务管理带来了新的挑战。文中提出分层结构下基于贝叶斯网络的网络服务故障定位算法。实验结果证明该算法具有周期性探针数目少、诊断率高、复杂度低的优点。

文中算法采取了以空间复杂度换取时间复杂度的方案,故障发生时能有效定位故障,但是在故障尚未发生时,需要根据服务依赖关系和网络结构更新知识库,而且网络和服务的自动修复机制使得服务和网络状态的获取变得更加复杂。下一步将引入时间窗口机制,使算法能更贴近实际应用。

#### 参考文献:

- [1] 凌晓东. SOA 综述[J]. 计算机应用与软件, 2007, 24(10): 122-124.

4 结束语

文中的方法和实验没有用到本体库的语义和层次关系,因为此次实验主要关注的是实例之间的冗余,可能冗余实例是跨概念和跨层次的,所以简单地对实例进行两两相似度的计算来找出冗余的实例。但是文中没有机械地将所有的实例进行相似度的计算,这样做是不可行的,因为数据太大。所以提出按照信息上的加权即属性出现的频数和属性字符串值的重复度来计算出一个可识别的属性集合,通过可识别的属性集合来得到可识别的实例集合。从实验结果看,它大大缩小了识别实例集合规模。由实验结果可以说明,文中的方法可以找出相似度较高的实例,是行之可效的;但是还是比较粗放,而且可识别实例集合还不是最优的。下面的工作是如何改进上面的方法使可识别实例集合最优并使集合实例间的重复度最大,并且将这些冗余的实例整合为唯一的实例。

参考文献:

[1] 滕广青,毕强.国外本体协调研究前沿进展及热点分析[J].中国图书馆学报,2012(1):113-120.

[2] 沈亦军,吕刚.基于实例相似度的本体映射方法研究[J].重庆科技学院学报(自然科学版),2012,14(3):170-172.

[3] Leacock C,Chodorow M. Combining local context and WordNet similarity for word sense and WordNet similarity for word

sense identification[M]//WordNet:an electronic lexical database. Cambridge,MA:MIT Press,1998.

[4] 李文超,杨妮妮.基于本体的语义相似性研究[J].科学技术与工程,2012,20(21):5328-5330.

[5] Elmeleegy H,Elmagarmid A K,Lee J. Leveraging query logs for schema mapping generation in U-MAP[C]//Proceedings of the ACM SIGMOD international conference on management of data. Athens,Greece:[s. n.],2011.

[6] 徐德智,黄旭.一种基于冗余消除的本体映射后处理方法[J].计算技术与自动化,2012,31(3):88-91.

[7] RDF教程[S/OL].2013-09-11. <http://w3school.com.cn/rdf/index.asp>.

[8] 李华,苏乐.基于关联规则的本体相似度的综合计算方法[J].计算机应用,2012,32(9):2472-2475.

[9] Buccella A,Cechich A,Gendarmi D,et al. Building a global normalized ontology for integrating geographic data sources[J].Computers & Geosciences,2011,37(7):893-916.

[10] 刘秀磊,廖建新,朱晓民,等.本体匹配中基于词义组合的词法分析算法[J].电子学报,2012,40(8):1624-1630.

[11] 曹泽文,钱杰,张维明,等.一种综合的概念相似度计算方法[J].计算机科学,2007,34(3):174-175.

[12] 刘宏哲.一种基于本体的句子相似度计算方法[J].计算机科学,2013,40(1):251-256.

[13] The DBLP computer science bibliography[S/OL].2013-09-11. <http://www.informatik.uni-trier.de/~ley/db>.

[14] Chodorow K,Diroff M.MongoDB权威指南[M].程显峰,译.北京:人民邮电出版社,2011.

(上接第10页)

[2] Natu M,Sethi A S. Active probing approach for fault localization in computer networks[C]//Proc of 4th IEEE/IFIP workshop on end-to-end monitoring techniques and services. [s. l.]:IEEE,2006:25-33.

[3] 褚灵伟,邹仕洪,程时端,等.概率和噪声环境下基于主动探针的Internet服务故障管理[J].中国科学:E辑,2008,38(10):1733-1746.

[4] Lin A. A model-based automated diagnosis algorithm[M]//Methodology and Tools in Knowledge-Based Systems. Berlin:Springer,1998.

[5] Huang Xiaohui, Zou Shihong, Wang Wendong, et al. Fault management for Internet services: modeling and algorithms [C]//Proc of IEEE international conference on communications. Istanbul:IEEE,2006:854-859.

[6] 张顺利,邱雪松,孟洛明.网络虚拟化环境下的服务故障诊断算法[J].软件学报,2012,23(10):2772-2782.

[7] Liao J,Zhang C,Li T,et al. A quasi-optimal probabilistic fault localization algorithm in communication networks[J].Chinese Journal of Electronics,2011,20(1):151-154.

[8] 李晶,朱敏.一种基于事件驱动的SOA故障疑似集选择算法[J].计算机应用与软件,2011,28(5):181-183.

[9] 杜晓丽,朱程荣,熊齐邦.一种基于依赖图的故障定位算法[J].计算机应用,2004,24(B12):67-69.

[10] 范贵生,虞慧群,陈丽琼,等.基于Petri网的服务组合故障诊断与处理[J].软件学报,2010,21(2):231-247.

[11] Bagchi S,Kar G,Hellerstein J. Dependency analysis in distributed systems using fault injection: application to problem determination in an e-commerce environment[C]//Proc of 12th international workshop on distributed systems: operations & management. [s. l.]:[s. n.],2001:15-17.

[12] 唐渊,金可音,周昆,等.Web服务失败分类法[J].湖南工业大学学报,2009,23(2):73-76.

[13] 刘丽,况晓辉,方兰,等.Web服务故障的分类方法[J].计算机系统应用,2010,19(8):258-263.

[14] Ide J S,Cozman F G,Ramos F T. Generating random Bayesian networks with constraints on induced width[C]//Proc of European conference on artificial intelligence. [s. l.]:[s. n.],2004:323-334.

# 基于分层模型的组合服务故障定位算法

作者：[崔鹏帅](#)，[朱浩洋](#)，[任浩](#)，[CUI Peng-shuai](#)，[ZHU Hao-yang](#)，[REN Hao](#)

作者单位：[国防科学技术大学 计算机学院, 湖南 长沙, 410073](#)

刊名：[计算机技术与发展](#)

英文刊名：[Computer Technology and Development](#)

年，卷(期)：2014 (9)

本文链接：[http://d.wanfangdata.com.cn/Periodical\\_wjfz201409002.aspx](http://d.wanfangdata.com.cn/Periodical_wjfz201409002.aspx)